UNCLASSIFIED

DEFENCE RESEARCH ESTABLISHMENT
CENTRE DE RECHERCHES POUR LA DÉFENSE
VALCARTIER

DREV - TN 1999-101

Unlimited Distribution/Distribution illimitée

Spécification d'un langage de requête pour un système adaptatif

par

M. Bélanger, L. Lamontagne, J. Berger et A.-C. Boury-Brisset*

September/septembre 1999

*APG Solutions et Technologies inc.

Approved by/approuvé par

Section Head/chef de section

18 oct. 1999

Date

SANS CLASSIFICATION

AVERTISSEMENT

Les renseignements ci-dessus sont la propriété de Sa Majesté et sont communiqués au destinataire, étant entendu qu'ils seront utilisés uniquement à des fins d'information et d'évaluation. Toute utilisation à des fins commerciales, notamment à des fins de fabrication, est interdite. La diffusion de la présente publication ou de son contenu à des tiers est interdite sans le consentement écrit préalable du MDN du Canada.

© Sa Majesté la reine, représentée par le ministre de la Défense nationale, 1999

100-07193/

ABSTRACT

This technical note presents a query language that was defined for an object-oriented data storage that may be used for an adaptive intelligent system based on a blackboard architecture.

RÉSUMÉ

Cette note technique présente un langage de requête pouvant être utilisé pour un système intelligent adaptatif ayant une architecture de type "blackboard".

TABLE DES MATIÈRES

1.0 INTRODUCTION 1

2.0 Création d'objets 3

2.1 Fonction pour définir une classe 3

2.2 Fonction pour créer une instance de classe 3

2.3 Fonction pour copier une instance de classe 4

3.0 Suppression d'objets 5

3.1 Fonction pour détruire une instance de classe 5

4.0 Consultation d'objets 6

4.1 Fonction pour accéder à la valeur d'un attribut 6

4.2 Fonction pour lire la valeur du n-ième élément d'un attribut multivalué 6

4.3 Fonction pour extraire une sous-séquence d'un attribut multivalué 7

5.0 Recherche sur les objets 8

5.1 Fonction pour rechercher les instances d'une classe donnée 8

5.2 Fonction pour rechercher les instances satisfaisant certaines contraintes 8

5.3 Fonction pour rechercher une instance satisfaisant certaines contraintes 10

5.4 Fonction pour rechercher les instances satisfaisant des contraintes parmi un ensemble d'instances . 10

6.0 Modification d'objets 12

6.1 Fonction pour ajouter des valeurs d'attributs 12

6.2 Fonction pour remplacer des valeurs d'attributs 12

6.3 Fonction pour copier la valeur d'un attribut dans un autre objet 13

6.4 Fonction pour supprimer une des valeurs d'un attribut multivalué 13

6.5 Fonction pour supprimer la valeur d'un attribut multivalué 14

6.6 Fonction pour ajouter un attribut de type lien 14

6.7 Fonction pour modifier un attribut de type lien 14

6.8 Fonction pour supprimer un attribut de type lien 15

7.0 Vérification de contraintes 16

7.1 Prédicat vérifiant l'existence d'instances satisfaisant certaines contraintes 16

7.2 Prédicat vérifiant des contraintes par toutes les instances spécifiées 16

7.3 Prédicat vérifiant l'existence d'une des valeurs d'un attribut multivalué 17

7.4 Prédicat vérifiant l'existence d'un lien entre deux objets 17

7.5 Fonction comparant de valeurs multivaluées 18

8.0 CONCLUSION 19

9.0 BIBLIOGRAPHIE 20

ANNEXE A - COMPARAISON DE GBB ET DE CLIPS 21

A.1 Structure des objets de GBB et de CLIPS 21

A.2 Création d'objets 22

A.3 Consultation d'objets 23

A.4 Modification d'objets 24

A.5 Recherche sur les objets 26

A.7 Suppression d'objets 27

TABLEAUX I à V

FICHE DE SYNTHÈSE

L'automatisation complète ou partielle de la gestion des ressources permettra au commandement aérien d'améliorer sa performance et de diminuer ses coûts dans le cadre d'opérations de maintien de la souveraineté de l'espace aérien canadien, d'opérations de contingence (locales ou internationales), et d'opérations d'aide aux sinistrés. Dans cette optique, des scientifiques du Centre de recherches pour la défense de Valcartier ont proposé de développer un système de planification dynamique de missions pour la gestion des ressources. Une étude des techniques d'intelligence artificielle est en cours pour concevoir un système intelligent capable de modifier son comportement en fonction d'un environnement dynamique et incertain. Ce système de base sera utilisé pour développer un système de planification capable de réagir en temps réel aux modifications du monde extérieur par le réajustement des plans en cours et à être exécutés.

Cette note technique présente un langage de requête pour un dépôt de données orienté-objet pouvant être utilisé pour un système intelligent adaptatif ayant une architecture "blackboard".

1.0 INTRODUCTION

Des travaux ont été entrepris au CRDV dans le but de créer un système intelligent de planification de missions en temps réel. Suite à l'étude de différents systèmes à base de connaissances (réfs. 1, 2, 3), un ensemble de concepts ont été sélectionnés dans le but de réaliser un système intelligent adaptatif pouvant être utilisé pour construire un système de planification dynamique des missions. Le système de base aura une architecture du tableau noir (communément appelé "blackboard") ayant des capacités de raisonnement dirigé par les buts et permettant l'exécution concurrente de sources de connaissances. Les sources de connaissances pourront accéder concurremment aux objets du dépôt de données ("blackboard"). Les objets pourront être composés d'attributs simples, multivalués ainsi que d'attributs de type liens. Un gestionnaire de communications permettra l'accès à d'autres bases de connaissances et de données.

Une des premières étapes réalisées dans le cadre de cette activité a été la spécification d'un langage d'accès aux objets du "blackboard".

Il faut se rappeler que le but ultime d'un dépôt de données orientée-objet est de pouvoir accéder à de l'information stockée sous forme d'objets. Un langage de requête pour un dépôt de données orientée-objet devrait donc, pour le moins, permettre de créer des objets, de les consulter, de les modifier, de faire des recherches sur les objets du dépôt de données, ainsi que de les détruire. Afin de définir un langage de requête supportant ces activités minimales, les langages de requête de deux outils de développement de systèmes à base de connaissances possédant des langages de requête relativement évolués ont été étudiés. Ces deux outils sont GBB et CLIPS. Après avoir comparé ces deux langages de requête (voir Annexe A), un nouveau langage de requête a été spécifié pour un système intelligent adaptatif.

Cette note technique présente le langage de requête qui a été développé pour un dépôt de données orientées-objet utilisé par un système intelligent adaptatif ayant une architecture de type "blackboard".

Ce travail a été exécuté au CRDV entre mars et mai 1996 dans le cadre de l'unité de travail 3aa15, planification dynamique et exécution.

2.0 Création d'objets

Les objets sont des instances de classes, il faut donc commencer par définir une classe pour pouvoir créer des objets. Cette section présente le langage de requête permettant de définir une classe et de créer des instances de cette classe.

2.1 Fonction pour définir une classe

La fonction `define-classe` définit une nouvelle classe. Cette fonction inclut des arguments précisant les attributs et les liens de cette classe, les espaces de travail ("blackboards") dans lesquels seront placées les instances de classes, et éventuellement les fonctions liées aux événements pour cette classe (cf. GBB).

Syntaxe : (`define-classe` *<nom-classe>* *<super-classe>* ....)

<nom-classe> : nom de la nouvelle classe à définir.

<super-classe>... : nom d'une ou de plusieurs classes dont la nouvelle classe héritera.

2.2 Fonction pour créer une instance de classe

La fonction `create-instance` crée et retourne une nouvelle instance de la classe spécifiée en argument.

Syntaxe : (`create-instance` *<nom-classe>* [*<nom-instance>*])[1]

---

[1] Des arguments placés entre crochets sont optionnels.

<nom-classe> : nom de la classe pour laquelle une nouvelle instance est requise.

<nom-instance> : nom de l'instance créée fourni optionnellement. Par défaut, le système génère un nom automatiquement.

2.3 Fonction pour copier une instance de classe

La fonction `copy-instance` permet de faire une copie d'une instance de classe. Cette fonction crée une nouvelle instance de la classe et ne copie que les caractéristiques spécifiées en attribut.

Syntaxe : (`copy-instance` {*<instance>* | *<nom-instance>*}
[*<nom-classe>*] *<att-to-copy>*)[2]

<instance> : instance à copier, pourra être le résultat d'une requête pour retrouver l'instance.

<nom-instance> : nom de l'instance à copier.

<nom-classe> : classe de la nouvelle instance (argument optionnel, par défaut c'est la même classe que c'est de l'instance à copier).

<att-to-copy> : nom des attributs et liens à copier.

---

[2] Les accolades indiquent qu'il faut faire un choix entre les arguments séparés par une barre verticale.

3.0 Suppression d'objets

Un langage de requête doit permettre de supprimer un ou plusieurs objet(s) lorsque cet(ces) objet(s) n'est (ne sont) plus utile à l'application. Cette section présente le langage de requête permettant de détuire une classe.

3.1 Fonction pour détruire une instance de classe

La fonction `delete-instance` détruit l'instance spécifiée en argument.

Syntaxe : (`delete-instance` {*<instance>* | *<nom-instance>*} [*<signale-event>*])

<instance> : instance à supprimer, pourra être le résultat d'une requête pour retrouver l'instance.

<nom-instance> : nom de l'instance à supprimer.

<signale-event> : argument booléen optionnel indiquant si un événement doit être émis.

4.0 Consultation d'objets

Cette section présente le langage de requête permettant de consulter les instances d'une classe.

4.1 Fonction pour accéder à la valeur d'un attribut

La fonction `get-slot-value` retourne la valeur de l'attribut d'une instance particulière.

Syntaxe : (`get-slot-value` {*<instance>* | *<nom-instance>*} *<nom-att>*)

<instance> : instance à consulter.

<nom-instance> : nom de l'instance à consulter.

<nom-att> : nom de l'attribut dont la valeur est désirée.

4.2 Fonction pour lire la valeur du n-ième élément d'un attribut multivalué

La fonction `get-nth` retourne la valeur du n-ième élément d'un attribut multivalué d'une instance particulière.

Syntaxe : (`get-nth` *<rang-n>* *<nom-att>* {*<instance>* | *<nom-instance>*})

<rang-n> : n-ième élément de l'attribut *<nom-att>* de l'instance *<nom-instance>*

<nom-att> : nom de l'attribut.

<instance> : instance à consulter.

<nom-instance> : nom de l'instance à consulter.

4.3 Fonction pour extraire une sous-séquence d'un attribut multivalué

La fonction `get-subseq` retourne une sous-séquence d'un attribut multivalué d'une instance particulière. Plus précisément, cette fonction retourne les valeurs du i-ième élément jusqu'au j-ième élément d'un attribut multivalué.

Syntaxe : (`get-subseq` *<rang-i>* *<rang-j>*)*<nom-att>*
{*<instance>* | *<nom-instance>*})

<rang-i> : i-ième élément de l'attribut *<nom-att>* de l'instance *<nom-instance>*

<rang-j> : j-ième élément de l'attribut *<nom-att>* de l'instance *<nom-instance>*

<nom-att> : nom de l'attribut.

<instance> : instance à consulter.

<nom-instance> : nom de l'instance à consulter.

5.0 Recherche sur les objets

Cette section présente le langage de requête permettant de retrouver un/des objet(s) ayant des caractéristiques particulières (ex. la valeur spécifique d'un attribut).

5.1 Fonction pour rechercher les instances d'une classe donnée

La fonction `get-instances` retourne toutes les instances d'une classe donnée.

Syntaxe : (`get-instances` *<nom-classe>* *<liste-blackboards>*)

<nom-classe> : nom de la classe.

<liste-blackboards> : le nom des blackboards dans lesquels se retrouve des instances de cette classe

5.2 Fonction pour rechercher les instances satisfaisant certaines contraintes

La fonction `find-instances` retourne toutes les instances satisfaisant certaines contraintes pour une classe donnée.

Syntaxe : (`find-instances` *<nom-classe>* *<liste-blackboards>* *<contraintes>*)

<nom-classe> : nom de la classe.

<liste-blackboards> : le nom des blackboards dans lesquels se retrouve des instances de cette classe

<contraintes> ::= <condition> |
(not <condition>) |
(or <condition>$^{+}$) |
(and <condition>$^{+}$)

<condition>	::=	{<comp-simple> | <comp-composee>}
		{<acces-att> | <val>}
		{<acces-att> | <val>}
		| <predicat-lien>
<acces-att>	::=	(**get-slot-value** <*nom-att*>) |
		<*nom-classe*>.<*nom-att*>
<comp-simple>	::=	= | < | <= | > | >= | member
<comp-composee>	::=	subset | overlaps
<val>	::=	<val-simple> | <val-composee>
<val-simple>	::=	valeur d'un attribut monovalué
<val-composee>	::=	valeur d'un attribut multivalué
		ou d'un lien (liste)
<predicat-lien>	::=	(**linked-with** [<*nom-instance*>]
		<*nom-lien*> <*val-lien*>)

member : la valeur d'attribut considérée appartient à l'ensemble de valeurs donné.

subset : les éléments de la première valeur d'attribut appartiennent à la seconde.

overlaps : l'intersection des deux ensembles d'attributs multivalués est non vide.

La fonction `get-slot-value` est utilisée ici sans nom d'instance associée puisqu'on fait une recherche d'instances.

Exemple1 :
```
(find-instances 'vehicule '(transport)
                (< (get-slot-value 'age) 5))
```
Cette requête cherche parmi les véhicules du "blackboard" transport ceux qui ont moins de 5 ans.

Exemple2 :

```
(find-instances 'parking '(transport)
     (and (= (get-slot-value 'categorie) 'luxe)
          (subset '(Ferrari-1 BMW-9)
                  (get-slot-value 'contient)))
```

Cette requête cherche le stationnement de luxe qui contient les véhicules Ferrari-1 et BMW-9.

La requête pour rechercher les stationnements qui contiennent des véhicules de la marque 'Porsche (dont je ne connais pas les instances existantes) s'exprimerait par :

```
(find-instances 'parking '(transport)
     (overlaps
          (get-slot-value 'contient)
          (find-instances 'vehicule '(transport)
               (= (get-slot-value 'marque) 'Porsche)))
```

où (`get-slot-value` 'contient) retourne la liste des vehicules contenues dans le parking considéré, (`find-instances` 'vehicule ...) retourne la liste des véhicules de marque Porsche, et (overlaps ...) permet de restreindre l'ensemble des stationnements à ceux qui contiennent au moins une Porsche (intersection des ensembles non vide).

5.3 Fonction pour rechercher une instance satisfaisant certaines contraintes

La fonction `find-instance` retourne la première instance de la classe <*nom-classe*> satisfaisant les contraintes.

Syntaxe : (`find-instance` <*nom-classe*> <*contraintes*>)

<*nom-classe*> : nom de la classe.

<*contraintes*> : voir sous-section 5.2

5.4 Fonction pour rechercher les instances satisfaisant des contraintes parmi un ensemble d'instances .

La fonction `filter-instances` retourne toutes les instances satisfaisant les contraintes parmi un ensemble d'instances .

Syntaxe : (`filter-instances` *<liste-instances>* [*<nom-classe>*] *<contraintes>*)

<liste-instances> : le nom des instances utilisées dans la recherche

<nom-classe> : le nom de la classe est facultatif, cet argument est utilisé pour restreindre la liste d'instances fournie à celles de cette classe.

<contraintes> : voir sous-section 5.2

Exemple :
```
(filter-instances '( parking-1 parking-2 parking-3)
        'parking (> (get-slot-value 'capacite) 500))
```
Dans ce cas, la précision du nom de la classe considérée (parking) est inutile.

6.0 Modification d'objets

Cette section présente le langage de requête permettant de modifier les instances d'une classe.

6.1 Fonction pour ajouter des valeurs d'attributs

La fonction `add-slot-value` permet de donner des valeurs à des attributs d'une instance particulière. La nouvelle valeur est ajoutée à la valeur existante pour cet attribut.
NB : on précise la position d'insertion *<rang-i>* de la nouvelle valeur (éléments ordonnés).

Syntaxe : (`add-slot-value` {*<instance>* | *<nom-instance>*}
<nom-att> *<nouvelle-valeur>* [*<rang-i>*])

<instance> : instance à modifier.

<nom-instance> : nom de l'instance à modifier.

<nom-att> : nom de l'attribut à modifier.

<nouvelle-valeurt> : valeur à donner à l'attibut.

<rang-i> : i-ième élément de l'attribut.

Exemple : `(add-slot-value 'vehicule-1 'couleur 'noir)`
; après avoir fait une peinture personnalisée (rouge et noir).

6.2 Fonction pour remplacer des valeurs d'attributs

La fonction `replace-slot-value` permet de remplacer les valeurs existantes d'attributs d'une instance particulière.

Syntaxe : (`replace-slot-value` {*<instance>* | *<nom-instance>*} *<nom-att>* *<nouvelle-valeur>*)

<instance> : instance à modifier.

<nom-instance> : nom de l'instance concernée par la modification.

<nom-att> : nom de l'attribut à modifier.

<nouvelle-valeur> : valeur remplaçante (mono ou multivaluée).

Exemple : `(replace-slot-value 'vehicule-1 'couleur 'rouge)`
; après avoir repeint vehicule-1 en rouge.

6.3 Fonction pour copier la valeur d'un attribut dans un autre objet

La fonction `copy-slot-value` permet de recopier chacun des attributs spécifiés

Syntaxe : (`copy-slot-value` *<from-instance>* *<new-instance>* *<nom-att>*)

<from-instance> : instance à partir de laquelle l'attribut est copié.

<new-instance> : instance destinataire de la copie.

<nom-att> : nom de l'attribut à copier.

6.4 Fonction pour supprimer une des valeurs d'un attribut multivalué

La fonction `remove-slot-value` permet d'effacer (de détruire) la valeur d'un élément d'un attribut multivalué d'une instance particulière.

Syntaxe : (`remove-slot-value` *<val-att>* *<nom-att>* {*<instance>* | *<nom-instance>*})

<val-att> : valeur de l'attribut à enlever.

<nom-att> : nom de l'attribut multivalué à modifier.

<instance> : instance à modifier.

<nom-instance> : nom de l'instance à modifier.

6.5 Fonction pour supprimer la valeur d'un attribut multivalué

La fonction `delete-slot-value` permet d'effacer (de détruire) la valeur existante d'un attribut multivalué d'une instance particulière.

Syntaxe : (`delete-slot-value` *<nom-att>* {*<instance>* | *<nom-instance>*})

<nom-att> : nom de l'attribut dont la valeur doit être effacée.

<instance> : instance à modifier.

<nom-instance> : nom de l'instance à modifier.

6.6 Fonction pour ajouter un attribut de type lien

La fonction `add-link` ajoute un ou des pointeurs vers la ou les instances spécifiées en argument *<liste-instances>*. Elle effectue également la mise à jour des liens inverses, c'est-à-dire l'ajout de pointeurs de chaque instance de *<liste-instances>* vers l'instance *<nom-instance>*.

Syntaxe : (`add-link` *<nom-lien>* {*<instance>* | *<nom-instance>*} *<liste-instances>*)

<nom-lien> : nom du lien à modifier.

<instance> : instance concernée par l'ajout d'un lien.

<nom-instance> : nom de l'instance concernée par l'ajout d'un lien.

<liste-instances> : une ou plusieurs instances (sous forme d'une liste à un ou plusieurs éléments).

6.7 Fonction pour modifier un attribut de type lien

Les instances liées à *<nom-instance>* par le lien *<nom-lien>* restent en place en utilisant cette fonction. Dans le cas où ce n'est pas souhaité, utiliser la fonction de remplacement décrite ci-dessous.

Syntaxe : (`replace-link` *<nom-lien>* {*<instance>* | *<nom-instance>*} *<liste-instances>*)

<nom-lien> : nom du lien à modifier.

<instance> : instance concernée par le remplacement de la valeur de lien.

<nom-instance> : nom de l'instance concernée par le remplacement de la valeur de lien.

<liste-instances> : une ou plusieurs instances (sous forme d'une liste à un ou plusieurs éléments).

Cette fonction remplace la valeur existante d'un lien (pointeur vers une ou plusieurs instances) par une autre valeur donnée dans *<liste-instances>*. Elle effectue la mise à jour des liens et liens inverses en deux étapes (1- suppression des liens existants, 2- ajout de nouvelles valeurs de lien).

6.8 Fonction pour supprimer un attribut de type lien

La fonction `delete-link` permet de détruire un attribut de type lien d'une instance particulière.

Syntaxe : (`delete-link` *<nom-lien>* {*<instance>* | *<nom-instance>*})

<nom-lien> : nom du lien à détruire.

<instance> : instance à modifier.

<nom-instance> : nom de l'instance à modifier.

7.0 Vérification de contraintes

Cette section présente le langage de requête permettant de vérifier certaines contraintes.

7.1 Prédicat vérifiant l'existence d'instances satisfaisant certaines contraintes

Le prédicat `exists-instances` teste, parmi un ensemble d'instances (donné explicitement, ou ensemble des instances de la classe *<nom-classe>*), s'il existe des instances vérifiant les contraintes spécifiées dans *<contraintes>*.

Syntaxe : (`exists-instances` {*<liste-instances>* | *<nom-classe>*}
<liste-blackboards> *<contraintes>*)

<liste-instances> : liste des instances

<nom-classe> : nom de la classe déterminant les instances.

<liste-blackboards> : le nom des blackboards dans lesquels se retrouve ces instances.

<contraintes> : voir sous-section 5.2

Ce prédicat correspond à :

(`non-nul` (`find-instances` *<nom-classe>* *<liste-blackboards>*
<contraintes>))

7.2 Prédicat vérifiant des contraintes par toutes les instances spécifiées

Le prédicat `for-all-instances` teste, parmi un ensemble d'instances (donné explicitement, ou ensemble des instances de la classe *<nom-classe>*), si toutes les instances vérifient les contraintes spécifiées dans *<contraintes>*.

Syntaxe : (`for-all-instances` {*<liste-instances>* | *<nom-classe>*}
<liste-blackboards> *<contraintes>*)

<liste-instances> : liste des instances

<nom-classe> : nom de la classe déterminant les instances.

<liste-blackboards> : le nom des blackboards dans lesquels se retrouve ces instances.

<contraintes> : voir sous-section 5.2

Ce prédicat correspond à :

(= *<liste-instances>* (filter-instances *<liste-instances>* *<contraintes>*))

Si le résultat retourné par filter-instances est la liste <liste-instances> alors les contraintes sont vérifiées par toutes les instances spécifiées.

7.3 Prédicat vérifiant l'existence d'une des valeurs d'un attribut multivalué

Le prédicat `member-slot` vérifie si une valeur fait partie des valeurs d'un attribut multivalué.

Syntaxe : (`member-slot` *<val-att>* *<nom-att>* {*<instance>* | *<nom-instance>*})

<val-att> : valeur dont l'existence sera vérifiée.

<nom-att> : nom de l'attribut multivalué.

<instance> : instance concernée.

<nom-instance> : nom de l'instance.

7.4 Prédicat vérifiant l'existence d'un lien entre deux objets

Le prédicat `linked-with` teste si l'instance *<nom-instance>* est lié par le lien *<nom-lien>* aux instances contenues dans *<val-lien>*. L'argument *<nom-instance>* est optionnel quand le prédicat est utilisé au sein des fonctions du type find-instances qui considèrent chacune des instances d'une classe tour à tour (cf. exemple 2).

Syntaxe : (`linked-with` [*<nom-instance>*] *<nom-link>* *<val-lien>*)

<nom-instance> : nom de l'instance.

<nom-link> : nom du lien à vérifier.

<val-lien> : nom des instances liées par le lien.

Exemple1 :
```
(find-instances 'vehicule '(transport)
                (linked-with 'a-cote-de '(Renault5-1)))
```
Cette requête cherche parmi les véhicules du "blackboard" 'transport ceux qui sont stationnés à côté du véhicule renault5-1.

La requête suivante est équivalente :
```
(find-instances 'vehicule '(transport)
  (includes (get-slot-value 'a-cote-de) '(Renault5-1)))
```

7.5 Fonction comparant de valeurs multivaluées

La fonction `subset` teste si un ensemble de valeurs est un sous-ensemble d'un autre, c'est-à-dire si tous les éléments du premier appartiennent au deuxième (aucune considération n'est accordée à l'ordre des éléments).

Syntaxe : (`subset` *<exp-multivaluee1>* *<exp-multivaluee-2>*)

<exp-multivaluee1> : ensemble1 de valeurs.

<exp-multivaluee2> : ensemble2 de valeurs.

8.0 CONCLUSION

Cette note technique a présenté un langage de requête pouvant être utilisé par un système intelligent adaptatif ayant une architecture "blackboard". Ce langage a été utilisé comme base pour l'implantation d'un système de planification dynamique de missions pour la gestion des ressources.

9.0 BIBLIOGRAPHIE

1. Bélanger, M., Berger, J., Lamontagne, L. et Boury-Brisset, A.-C., "Architecture de l'outil GBB", CRDV TM-9730, UNCLASSIFIED.
2. Lamontagne, L., Bélanger, M., Berger, J., et Pelletier, P., "Évaluation d'un système à raisonnement procédural (UM-PRS)", CRDV TM-9738, UNCLASSIFIED.
3. Bélanger, M., Lamontagne, L., Berger, J., et Boury-Brisset, A.-C., "Architecture de l'outil ATOME", CRDV TM-9731, UNCLASSIFIED.
4. Lyndon B. Jonhnson Space Center, "CLIPS Reference Manual", CLIPS Version 6.0, JSC-25012, June 2nd, 1993.
5. Lyndon B. Jonhnson Space Center, "CLIPS Architecture Manual", CLIPS Version 5.1, JSC-25014, January 6th, 1992.

ANNEXE A

COMPARAISON DE GBB ET DE CLIPS

Cette annexe présente brièvement la structure des objets de GBB et de CLIPS ainsi que leur langage de requête.

A.1 Structure des objets de GBB et de CLIPS

La structure des objets de GBB (réf. 1) est constituée d'attributs simples ainsi que d'attributs de type lien. La notion de liens permet d'établir une relation entre les instances d'une même classe ou de classes différentes, celles-ci pouvant appartenir au même espace ou à des espaces différents. Un lien est en fait un attribut d'instance mettant en relation plusieurs instances. Les liens sont bidirectionnels ("outcoming" et "incoming") et peuvent être de type simple ou multiple.

La structure des objets de CLIPS (réfs. 4 et 5) est constituée d'attributs simples ainsi que d'attributs multivalués. Un attribut multivalué est un ensemble ou une liste ordonnée d'éléments de type homogène. Le concepteur d'une application donnera une sémantique à chacune de ces valeurs.

A.2 Création d'objets

Le tableau I montre les fonctions de CLIPS et de GBB permettant la définition de classes ainsi que la création et l'initialisation d'objets.

On remarque que dans CLIPS aussi bien que dans GBB, les instances peuvent être créées à partir d'une classe ou à partir d'un autre objet par duplication.

TABLEAU I

Fonctions pour la création d'objets

	CLIPS	GBB
Définition de classes		
	fonction `defclass-construct`	fonction `define-unit-class`
Création d'instances		
	fonction `make-instance`	fonction `make-unit` (une instance)
	fonction `definstances`	
	fonction `duplicate-instance`	`copy-unit` (utilise copy-unit-slot)
Initialisation d'instances		
	fonction `initialize-instance`	fonction `make-unit` (une instance)

A.3 Consultation d'objets

Le tableau II montre les fonctions de CLIPS et de GBB permettant la consultation d'objets. La consultation d'un objet se résume principalement à la lecture d'une valeur d'un attribut. Puisque CLIPS permet les attributs multivalués, il fournit la fonction permettant d'avoir accès directement à un champs précis d'un attribut multivalué.

TABLEAU II

Fonction de consultation d'objets

	CLIPS	GBB
Lecture de la valeur d'un attribut		
	méthode `get-<slot>`	unit-slot-value ou accesseur classe.slot
	fonction `nth$` pour la lecture d'un champ multivalué	

A.4 Modification d'objets

Le tableau III montre les fonctions de CLIPS et de GBB permettant la modification d'un ou de plusieurs objets.

Tout dépendant de la structure interne des objets, on peut voir que les fonctions de modification seront différentes. La possibilité d'avoir des attributs multivalués ou des liens demande des fonctions pour la manipulation de ces structures particulières. CLIPS a la possibilité de modifier d'une seule fonction, plusieurs objets.

TABLEAU III

Fonctions de modification d'objets

	CLIPS	GBB
Modification de la valeur d'un attribut		
	méthode put-<slot> (un attribut)	setf + unit-slot-value
	modify-instance (plusieurs)	
	active-modify-instance	
	message-modify-instance	
	active-message-modify-instance	
	Permet manipulation attributs multivalués	
Modification de liens:		
		Ajout : linkf
		remplace : linkf !
		annule : unlinkf
Action sur les instances		
	do-for-instance	
	do-for-all-instances	
	delayed-do-for-all-instances	

A.5 Recherche sur les objets

Le tableau IV montre les fonctions de CLIPS et de GBB permettant la recherche d'objets.

TABLEAU IV

Fonctions de recherche sur les objets

	CLIPS	GBB
Tests sur attributs		
	slot-exists-p, ...	idem (fonctions CLOS)
recherche d'une instance		
		fonction `get-units`
recherches d'instances		
	Find-all-instances	Find-units (toutes les instances satisfaisant des contraintes)
	any-instance-p	
	find-instance (la première)	(car (find-units ...))
		Filter-units (à partir d'un ens. d'instances)
	pattern-matching	

A.7 Suppression d'objets

Le tableau V montre les fonctions de CLIPS et de GBB permettant la suppression d'objets. CLIPS permet la suppression de plusieurs objets simultanément alors que GBB ne permet que la suppression d'un objet à la fois.

TABLEAU V

Fonctions de suppression d'objets

	CLIPS	GBB
Suppression d'instances		
	méthode `delete` (une seule instance)	fonction `delete-unit`
	fonction `unmake-instance` (supprime plusieurs instances avec delete)	